Flash-MoE: Run a 397B Model on a 48GB Laptop (Here's How)

March 22, 2026 · by Mark Bartlett

Download this guide as PDF

> **Quick Answer:** Flash-MoE is a pure C/Metal inference engine that runs the 397-billion-parameter Qwen3.5-A17B on a MacBook Pro M3 Max with 48GB RAM at 4.4 tokens/second. It streams 209GB of expert weights from your SSD on demand, keeping only 5.5GB resident in memory. The 4-bit config is production-usable with tool calling support. The 2-bit config hits 5.7 tok/s but breaks JSON output. You need an M3 Max or better, 48GB unified memory, and a 1TB SSD. Mac-only. No Linux, no Windows.

📚 **More on this topic:** MoE Models Explained · Qwen3 Complete Guide · VRAM Requirements · Apple Silicon Local AI · Autoresearch Guide

A 397-billion-parameter model. On a laptop. At conversational speed.

That's the claim behind Flash-MoE, a project by Dan Woods that runs Qwen3.5-397B-A17B on a MacBook Pro M3 Max with 48GB of unified memory. The model is 209GB on disk. The engine uses 5.5GB of RAM. The rest streams from your SSD, on demand, at 4.4 tokens per second.

If your first reaction is skepticism, good. The numbers deserve scrutiny. But the code is public, the technique works, and if you're running local AI on Apple Silicon, you should understand what's happening here.

What Flash-MoE actually is

Flash-MoE is a from-scratch inference engine written in C, Objective-C, and Metal shaders. No Python, no PyTorch, no MLX. About 7,000 lines of C/Obj-C and 1,200 lines of Metal GPU kernels.

It runs one specific model: Qwen3.5-397B-A17B, a Mixture-of-Experts model with 397 billion total parameters but only 17 billion active per token. The model has 512 experts per layer, and Flash-MoE activates just 4 of them (plus one shared expert) for each token.

The whole trick is that 512 experts don't need to live in RAM. They live on your SSD. The engine loads the 4 it needs, runs them, and moves on.

Component	Size	Where it lives
Non-expert weights (embeddings, routing, attention)	5.5GB	RAM (memory-mapped)
Metal scratch buffers	~200MB	RAM
Expert weights (4-bit)	209GB	SSD, streamed on demand
Expert weights (2-bit)	120GB	SSD, streamed on demand

That 5.5GB resident footprint leaves 42GB of your 48GB unified memory for the OS and page cache. The page cache is doing real work here — frequently-used experts stay cached, hitting about 71% of the time.

How SSD expert streaming works

The idea comes from Apple's 2023 research paper "LLM in a Flash," which proposed using flash storage to run models larger than available memory. Flash-MoE takes that concept and builds a production engine around it.

For each layer of the model, the pipeline looks like this:

1. GPU runs attention and routing, figures out which 4 experts this token needs (1.22ms)
2. CPU dispatches parallel SSD reads via `pread()` and Grand Central Dispatch, loading the 4 expert weight files, ~6.75MB each (2.41ms)
3. GPU runs the expert forward pass, dequantizes, computes, fuses results back into the residual stream (0.04ms, deferred to overlap with next layer)

Average per-layer latency: 4.28ms at 4-bit. Across 60 layers, that's your ~4.4 tok/s.

Here's something counterintuitive: on Apple Silicon, SSD reads and GPU compute share the same memory controller. You can't overlap them profitably. The serial pipeline (GPU, then SSD, then GPU) is actually hardware-optimal, even though it looks like you're leaving performance on the table.

The Metal shaders include a custom FMA-optimized dequantization kernel that rearranges the 4-bit math from `(nibble * scale + bias) * x` to `fma(nibble, scale*x, bias*x)`. Pre-computing `scale*x` and `bias*x` lets the GPU use a single fused multiply-add instruction. That optimization alone accounts for a 12% speedup.

Performance numbers

Config	Tokens/sec	Disk size	Quality
4-bit (production)	4.36	209GB	Full quality, tool calling works
4-bit (warm cache)	4.8	209GB	After page cache fills
2-bit	5.74	120GB	Breaks JSON, unreliable tool calling
2-bit (peak single token)	7.05	120GB	Burst, not sustained

The 4-bit config is the one that matters. 4.36 tok/s is slow compared to running a 7B model at 60+ tok/s, but this is a 397B-parameter model on a laptop. For comparison, Ollama running Qwen3-32B on the same hardware does about 25 tok/s. You're trading speed for model scale.

An M5 Pro user on Hacker News reported 6.55 tok/s at 4-bit, which tracks with the faster SSD and memory bandwidth on newer chips. This technique scales directly with SSD speed.

The 2-bit trap

The 2-bit quantization gets the headline-friendly 5.7 tok/s number, but it breaks things. The repo's own issue tracker documents it: 2-bit produces `\name\` instead of `"name"` in JSON output, making tool calling and structured output unreliable.

Simon Willison flagged another concern: the project claims 2-bit output quality is "indistinguishable from 4-bit," but the evaluation methodology behind that claim is thin. No standardized benchmarks, no MMLU, no GPQA scores for the 2-bit config.

For context, tarruda (the Neovim creator) ran Qwen 3.5 397B at ~2.5 BPW on an M1 Ultra with 192GB and hit 20 tok/s with MMLU 87.86% and GPQA 82.32%. But that's a $7,000 machine with 4x the memory. Flash-MoE does it on $3,500 hardware with 48GB.

How it was built

Woods used Claude Code with Andrej Karpathy's autoresearch pattern, the "run experiments while you sleep" approach that hit 29,000 GitHub stars in a week. 90 experiments to optimize the Metal kernels and streaming pipeline. Claude Opus 4.6 authored most of the resulting research paper.

The codebase is small for what it does: `infer.m` (the full engine, ~7,000 lines), `shaders.metal` (GPU kernels, ~1,200 lines), `chat.m` (interactive TUI with tool calling), and `tokenizer.h` (a 449-line single-header BPE tokenizer in C).

This is an AI-built inference engine running an AI model. Make of that what you will.

Hardware requirements

Requirement	Minimum	Recommended
Chip	M3 Max	M4 Max / M5 Pro
Unified memory	48GB	48GB+
SSD	1TB NVMe	1TB+ (faster = faster inference)
macOS	26.2+	Latest
Free disk space	215GB (4-bit)	215GB

The 48GB unified memory requirement is firm. The engine needs 5.5GB resident plus enough headroom for the OS page cache to be useful. With 36GB or less, cache hit rates would tank and you'd be reading from SSD on nearly every expert load.

This is a $3,000+ MacBook. One Hacker News commenter put it well: "I'm getting tired of 'laptop' in every one of these clickbait titles turning out to be a $3,000 MacBook." Fair point. But the M3 Max with 48GB is the cheapest hardware that can run a 397B model at conversational speed, period. The next cheapest option is a desktop with multiple GPUs.

Linux and Windows users: this doesn't work for you. The SSD streaming technique could work on Linux with io_uring in theory, but nobody's built it yet.

What's wrong with it

It's Mac-only. The entire engine is built on Metal and Apple's memory architecture. No CUDA port, no ROCm port.

It runs one model. Flash-MoE is hardcoded for Qwen3.5-397B-A17B. The weights are baked into the engine's expectations. This is a research project, not a general-purpose inference server.

It uses 4 experts instead of 10. The standard Qwen3.5 config activates 10 experts per token. Flash-MoE drops that to 4 to fit the SSD bandwidth budget. Woods reports the “biggest quality drop-off occurred at 3,” but going from 10 to 4 is still a real quality hit. One Hacker News commenter called this “particularly misleading” when combined with aggressive quantization.

It’s early-stage software. The repo has open issues, the chat interface is a TUI, and there’s no OpenAI-compatible API server. You can’t plug this into Open WebUI or any existing chat frontend.

Your SSD speed is your inference speed. An older Mac with a slower NVMe will be noticeably slower. SSD reads don’t cause wear (only writes do), but you are reading 209GB repeatedly, so your SSD controller is working hard.

And a well-tuned 30B at 4-bit might just be more useful. Several commenters pointed out that for practical work, a 30B model that fits entirely in RAM at 25+ tok/s may produce better results than a 397B model at aggressive quantization with reduced experts. The parameter count is impressive. Parameters aren’t everything.

Why you should care

Every local AI guide (including ours) frames model selection around one question: will it fit in RAM? Flash-MoE changes that. If your model is MoE and your storage is fast enough, you can stream expert weights on demand and run models far larger than your memory.

The Apple “LLM in a Flash” paper came out in 2023. Flash-MoE is the first project to build a complete, usable engine around the idea. The 71% page cache hit rate suggests that smarter prefetching could push speeds higher. And the SSD streaming concept isn’t Mac-specific in principle, even if this implementation is.

For now, it’s a proof of concept with real utility for Mac users who want to run the largest open model available. 4.4 tok/s at 397B parameters, from a laptop SSD. Six months ago, nobody was doing that.

How to try it

```
# Clone the repo
git clone https://github.com/danveloper/flash-moe.git
cd flash-moe/metal_infer
```

```
# Build
make

# Download model weights (209GB for 4-bit — this will take a while)
# Follow the repo's weight download instructions

# Run inference
./infer --prompt "Explain quantum computing" --tokens 100

# Interactive chat with tool calling
./chat

# Timing breakdown (useful for benchmarking your hardware)
./infer --prompt "Hello" --tokens 20 --timing

# 2-bit mode (faster, but broken JSON — not recommended for production)
./infer --prompt "Explain quantum computing" --tokens 100 --2bit
```

Check the repo for weight download instructions. The 209GB expert weights go in a `packed_experts/` directory, and the 5.5GB non-expert weights go in `model_weights.bin`.

Related guides

- MoE Models Explained: Why Fewer Active Parameters Means Faster Inference
- Qwen3 Complete Guide: Every Model from 0.6B to 235B
- How Much VRAM Do You Need for Local LLMs?
- Running LLMs on Mac M-Series
- How to Run Karpathy's Autoresearch on Your Local GPU

Get notified when we publish new guides.

Subscribe — free, no spam

Source: https://insiderllm.com/guides/flash-moe-run-397b-model-laptop/

Free guides for running AI locally